FACULTAT
DE CIÈNCIES
I TECNOLOGIA
UVIC | UVIC·UCC

UNIVERSITAT DE VIC

TREBALL FINAL DE GRAU

GRAU EN ENGINYERIA MECATRÒNICA

Model and build Robot in ROS environment

Autor:
Javier Rubia Estruch

Tutors:
Dr. Andreu Corominas Murtra
Dr. Moisès Serra-Serra

June 8, 2020

Acknowledgement

I would like to express my deep and sincere gratitude to my project supervisors, Dr. Andreu Corominas Murtra and Dr. Moisès Serra Serra, and the robotics team of the University of Vic, for giving me the opportunity to do this project and providing invaluable guidance throughout it.
Andreu Corominas has carried me with the adventure of ROS with endless patience and always with friendly attitude which I'm thankful for.
Moisès Serra I appreciate the interest and the motivation provided at any time and especially in this times where the situation is more complicated than ever.
Also, I'm grateful with my family for all the support and a special thanks to my father who has been always there fighting with me in silence.
Finally to my Ita the love of my life.

Abstract

Bring a robot into ROS environment could be something difficult, but the idea that have some hardware and software so well integrated that can be even simulated with the help of some tools that ROS provides is really interesting, the intention of this project is to design, model and build a mobile robot with holonomic properties that will be able to compete in robotics competitions in the future, also have the capability to be teleoperated, and creating a brand new and standardized ROS packages to control it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This project is born due to the collaboration with the Robotics team from University of Vic and the opportunity to make a step further in the evolution of the team to establish a great programming base with a simulation, that will improve the movements of the robot in further competitions.

Personally, I'm a huge fan of robots and programming, during my membership of 3 years in the robotics team of the university every year the idea of bringing the team's robot *Garrinator* to a ROS system, it's been a huge challenge that never became real, but this year I've had the opportunity to have a closer look to the common and complex algorithms that runs normally in mobile robots. Even that is my first time in this field I felt really motivated against this challenge to give a proper ROS environment and awaiting greater progress in this path on further competitions.

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

The robot used in this project is a class of mobile robot which uses omnidirectional wheels that gives it plenty of mobility in all ground directions.

The main characteristics of the robot are:

- ROS based system
- 4 Omni wheel configuration
- Brushless with encoder powertrain.

From now on, the robot of this project will be referred as *Garrinator*.

1.1 Objectives

The main goal of this project is to design , model and build a robust robot that will be easily adaptable to the new challenges in further robotic competitions. In order to achieve that goal the project has the following objectives:

- Design the base of the robot that has a power train with 4 omnidirectional wheels, powered with brushless motors.
- Model and program the robot using the Operating System ROS and C++.
- Capability of the robot to be teleoperated.

1.2 Structure of the memory

The following document follows a specific order. Each chapter includes a part of the project:

- Chapter 1 Introduction
- Chapter 2 State of the art
- Chapter 3 Omni wheels kinematics
- chapter 4 Hardware architecture
- chapter 5 Software architecture
- chapter 6 Conclusions
- chapter 7 Results

To conclude this chapter, this project has an ambitious goal that will require a lot of self learning, because the university do not provide any subject where the basics of ROS are studied, the ROS concepts that are studied where theoretical basics of odometry, kinematics and deeper knowledge of localization and mapping algorithms. In addition is the first time that the robotics team and myself work with the ODrive motor driver, so a technical research has been carried out to know how it works.

Chapter 2

State of the art

This chapter will cover a comparison with the robots that are actually available in the market or in development. The main characteristic among them is that they are built over a ROS system. This chapter will compare the pros and cons with the *Garrinator*. Another selection criteria is that they must have holonomic movement and the dimensions have to be between 0.2 X 0.2 X 0.2m and 1 X 1 x 1m .
The robots are the following: *Nav2* a multi purpose mobile platform, *Seekur* holonomic indoor/outdoor payload transporter, *X-terrabot* a holonomic mobile platform with a robotic arm, *Mecanumbot* a mobile platform used to practice and develop ROS concepts.
The next pages will give more details about the principal characteristics of the selected robots.

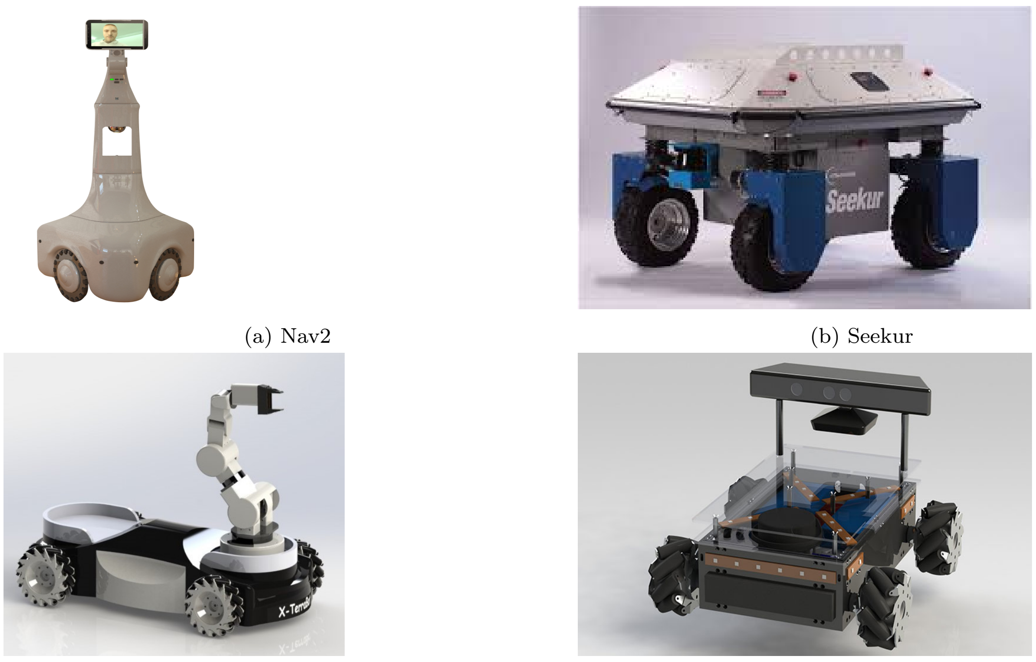

(a) Nav2 (b) Seekur (c) X-terrabot (d) Mecanumbot

Figure 2.1: Robots to compare with *Garrinator*

- Nav2
  - Purpose:
    The Nav2 is a highly modular and customizable robotic platform, some of its applications are: Telehealth, Telepresence, automated pill dispenser.
  - Especifications:
    * **Brushless motors**
    * **3 Omni directional wheels**
    * 360º Vision system with video transmission
    * Touchscreen-based device (tablet, smartphone)
    * Obstacle avoidance
    * Speakers and microphone
    * Localization and navigation.
  - Pros/con/differences:
    The nav 2 platform has an easily human-robot interface as it has monitor, microphone and speakers, also it has 360º of vision, but it has almost no load space, the main difference between them is field of application; the nav 2 is telecommunication or health supporter, while the *Garrinator* is development and robotics competitions.
- X-Terrabot
  - Purpose:
    X-Terrabot is a highly multi-purpose robot, some work fields of this robot are: transport in factories or warehouses,hazardous material handling and military applications. Currently is still a research product.
  - Main characteristics:
    * 6 Axis robotic arm with a gripper.
    * **4 Mecanum wheels**.
    * Localization and navigation.
  - Pros/cons/differences:
    Both robots has research purposes that can end up to a very useful robot, the main difference between them is the X-terrabot has navigation included and 4 mecanum wheels.

- Seekur
  - Purpose:
    Seekur has various types of application depending on the hardware options, one example is to transport, indoor/outdoor medium payloads, or just be the mobile base of another application.
  - Main characteristics:
    - Dimensions: 1,4 × 1,3 × 1,1 m
    - Weight: 300 kg
    - **4 standard wheels with 4 omnidirectional drivers**
    - Mono and stereo vision cameras
    - GPS
    - 6 DOF Inertial Mesurement Unit.
    - Obstacle avoidance
    - Navigation and path planning.
  - Pros/con/differences This is a big outdoor focused robot. Its weight is 300kg and it is really robust and tough, it has much more payload than the *Garrinator*.

    But its manoeuvrability can be drastically reduced in small spaces as it is much bigger.
- Mecanumbot
  - Purpose:
    Mecanumbot is designed as a software research platform, so basically it's a developmental robot that can allow anyone who's learning ROS and wants to build a robot try new algorithms and methods of localization.
  - Main characteristics:
    - Motors: Brushed of 2,5A.
    - **4 Mecanum wheels** of 100mm of diameter.
    - Microsoft Kineckt
    - Neato Laser Scanner
    - Obstacle avoidance
    - Localization and navigation.
  - Pros/con/differences:
    Both Robots are for research purposes, the main differences between them is the camera, the type of the wheels and the localization and navigation capabilities.

To summarize the chapter, the *Garrinator* is not common at all in factories or transport duties, the fact that has 4 omnidirectional wheels do not make it unique as there are more robots that implement that technology.
In the other hand it has a notable disadvantage against the robot that uses 4 mecanum wheels, that is because when moving forward/backward the *Garrinator* uses the power of 2.85 motors, but the mecanum ones use the power of 4 since all wheels are steered in the direction of motion.

Chapter 3

Omniwheel kinematics

This chapter will introduce some concepts of mobile robotics, kinematics and the structure of the *Garrinator* with 4 omni wheels. It is based on the grade course "Mobile Robotics", with help of *Some Essential Notes for Robotics* from *Andreu Corominas-Murtra* .
In order to be able to find the inverse kinematics matrix of the robot, some previous knowledge will be needed. The first subsection will introduce the *Coriolis Law* which can relate the movement of an object referred to a fixed frame, in our case could be a robot (fixed frame) and wheels (mobile frame), also the concept of *twist* which is important to understand in the future sections.

3.1 Coriolis Law

Given two coordinate frames, one called moving F^M which is rotating at an angular velocity $\mathbf{w}$ about the other frame, called fixed F^F. Then the derivative of any vector $\mathbf{x}$ with respect to the time is related by:

$$\frac{d\mathbf{x}^F}{dt} = \frac{d\mathbf{x}^M}{dt} + \mathbf{w} \times \mathbf{x}^F \tag{3.1}$$

Where the expression $\mathbf{w} \times \mathbf{x}^F$ is the cross vector product. The $\mathbf{x}$ can be, for instance, position, velocity or force.
In the practical situation of a robot rotating about a fixed frame of reference, a given point p of that robot (fixed with respect to the vehicle frame) has a null time derivative in the vehicle frame ($\frac{d\mathbf{p}^M}{dt} = 0$), so equation 3.1 simplifies to the expression of the linear velocity of any point of the vehicle according to the rotation of the vehicle and the position of that point with respect to a fixed frame.

$$v = \frac{d\mathbf{p}^F}{dt} = \mathbf{w} \times \mathbf{p}^F \tag{3.2}$$

3.2 Twist

The *twist* is the expression of the velocity of a rigid body. It is composed by two vectors which express linear and rotational velocities of the body. For instance a robot that could move to any direction in the 3D space, has a *twist* such as:

Linear component velocity: $\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$

Angular component velocity: $\begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}$

Final 3D *twist*: $\begin{bmatrix} v_x \\ v_y \\ v_z \\ w_x \\ w_y \\ w_z \end{bmatrix}$

If the movements of the robot are restricted on a 2D plane, as the robot won't go uppers in the air or below the ground, the $v_z = 0$. Regarding rotations the only non-null rotation is the w_z (arround he vertial azis). Therefore, the 2D *twist* looks like:
2D *twist*: $\begin{bmatrix} v_x \\ v_y \\ w_Z \end{bmatrix}$
This *twist* will describe how the robot moves through the floor.

3.3 Kinematics

Robot kinematics is the study of a robot's motion without considering the cause of it. There are two types of kinematics equations: forward and inverse. Kinematics equations vary by the type of robot and actuator position/configuration.

3.3.1 Forward kinematics

The forward kinematic equations of a robot will give the twist of a given frame of interest of the robot, by knowing each of the wheel rotation velocities.

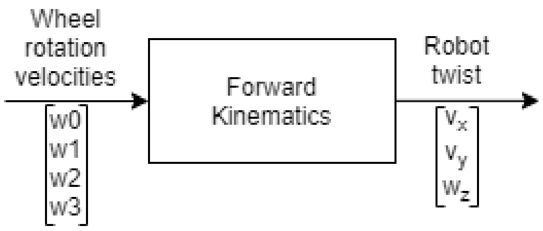


Figure 3.1: Forward Kinematics

3.3.2 Inverse kinematics

The inverse kinematic equations are capable to transform a robot *twist* into the needed wheel velocities for each wheel in order to achieve the desired robot movement.

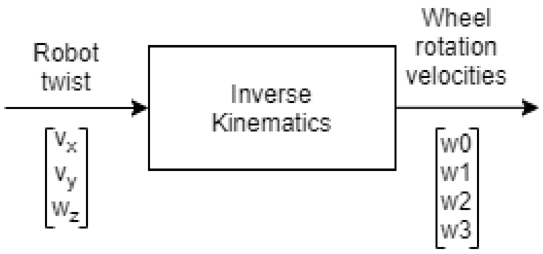


Figure 3.2: Inverse Kinematics

3.4 4 omni wheel inverse kinematics

This section will cover the inverse kinematics applied to a 4 omni wheeled robot, this will be a relation between the wheel movement and the state of the robot, which is a 2D*twist*. The following equations are generic for omni wheels robot, leading to a generic expression with independently of the number of wheels and their position. Finally the equations will be applied to the *Garrinator* wheel configuration.
In order to obtain the inverse kinematics matrix the first step is to find the linear velocity of the i-th wheel center with respect to the robot base frame F^R, by applying the Coriolis law:

$$v_i^R = \begin{bmatrix} v_x \\ v_y \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ w_z \end{bmatrix} \times \begin{bmatrix} x_i \\ y_i \\ 0 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ 0 \end{bmatrix} + \begin{bmatrix} -w_z y_i \\ w_z x_i \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -y_i \\ 0 & 1 & x_i \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} \tag{3.3}$$

Once the linear velocity of the wheel center is computed, the wheel center linear velocity with respect to the vehicle frame, it is possible to transform it to the wheel frame by applying the rotation on the 2D plane.

$$v_i^w = \begin{bmatrix} cos\beta_i & sin\beta_i \\ -sin\beta_i & cos\beta_i \end{bmatrix} \begin{bmatrix} 1 & 0 & -y_i \\ 0 & 1 & x_i \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} \tag{3.4}$$

Finally, from v_i^w, it's only needed the first component u_i, which is the actuated velocity (aligned with the wheel driving direction):

$$u_i = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} cos\beta_i & sin\beta_i \\ -sin\beta_i & cos\beta_i \end{bmatrix} \begin{bmatrix} 1 & 0 & -y_i \\ 0 & 1 & x_i \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} \tag{3.5}$$

Which can be simplified to:

$$u_i = \begin{bmatrix} cos\beta_i & sin\beta_1 & -y_i cos\beta_i + x_i sin\beta_i \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} \tag{3.6}$$

The image below describes the configuration of the *Garrinator* wheels, the wheels are at 150mm from the center of the robot, and all of them are angularly separated 90º.

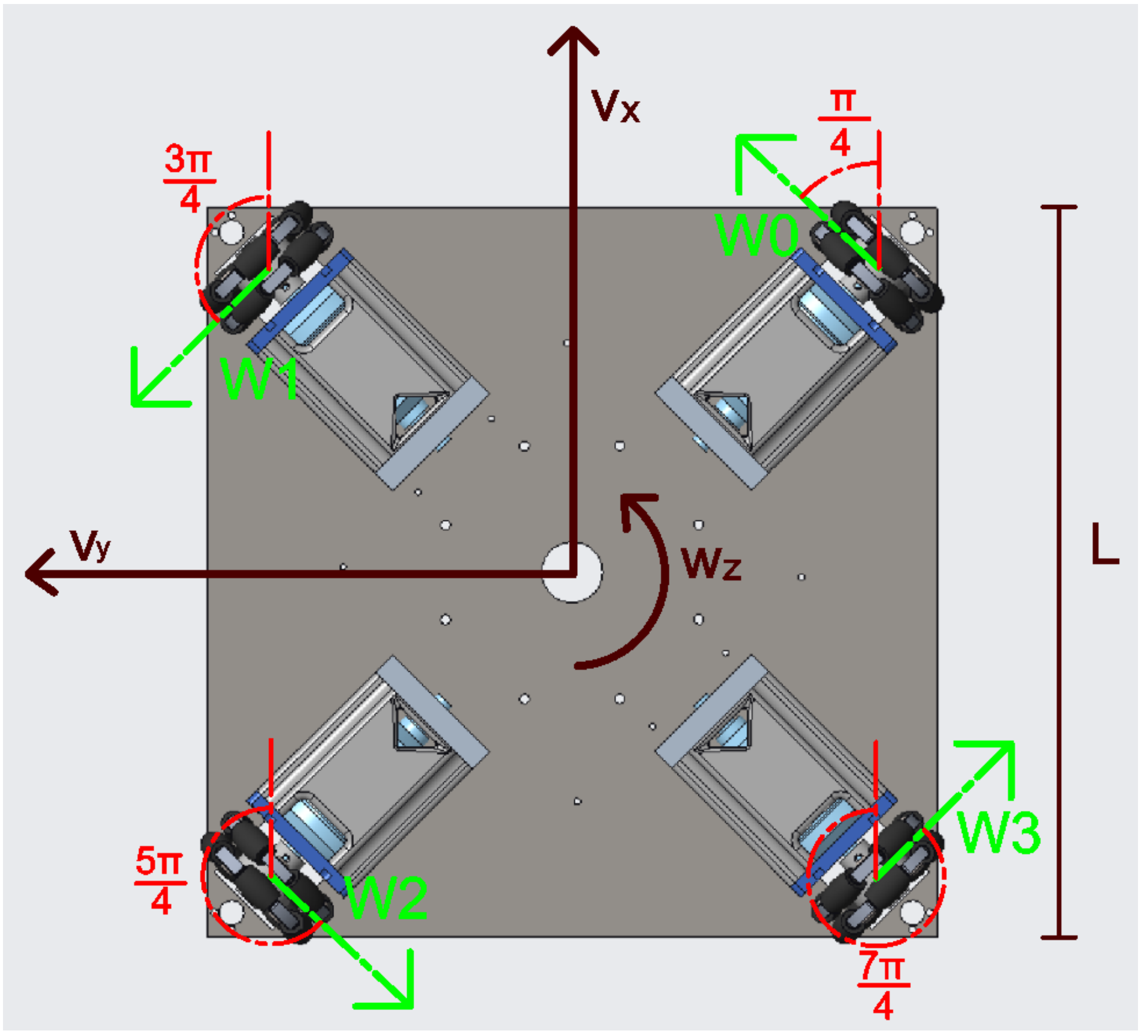


Figure 3.3: *Garrinator* omni wheels configuration, with β_i angles.

The dimensions referred to the configuration of the wheels are the following:

	x_i	y_i	β_i
w0	150mm	-150mm	$\frac{\pi}{4}$
w1	150mm	150mm	$\frac{3\pi}{4}$
w2	-150mm	150mm	$\frac{5\pi}{4}$
w3	-150mm	-150mm	$\frac{7\pi}{4}$

Table 3.1: Table of the Kinemaatics constructive parameters for the *Garrinator* robot.

Taking the equation 3.6 and combining the table 3.1 with the wheel distances and angles, it leads to the full inverse kinematics:

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{0.3}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{0.3}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & \frac{0.3}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & \frac{0.3}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix} \tag{3.7}$$

With this inverse kinematics the system is able to know the amount of angular velocity (rad/s) that will need each wheel when a 2D*twist* is applied to the robot. To conclude this chapter, the equation 3.7 will be really powerful in order to move the robot in the application.

Chapter 4

Hardware architecture

This chapter will introduce all the hardware components that are part of the *Garrinator* and how they are connected, first of all will give some important specifications of the components, then the main configuration and how they work are specified below.

Robot specifications :

- Dimensions: 300 x 300 x 100 mm
- Weight: 5 kg aprox.
- Wheels: 4 **Omni wheels** 60mm diameter with 10 rollers.
- Motors: 4 **Brushless motors** D5065 270KV max. current 65A.
- Max speed: 2 m/s.
- Motor driver: 2 **ODrive V3.6** Electronic Speed Control 120A max peak current.
- Power supply: Battery type **LiPo** 4s (14.8V) **5200mAh 50C** (max. discharge 260A).
- Joystick gamepad to teleoperate the robot.

Computer specifications:

- Model: **NUC7i7DNH**.
- Processor: Intel® Core™ i7-7567U processor (3.5 GHz/4.0GHz Turbo).
- Disk drive: Corsair M.2 **240 Gb SSD**.
- RAM: Crucial **8GB DDR4**.
- Operating system: Linux **Ubuntu 18.04** LTS.
- ROS version: **Melodic Morenia**.

4.1 What is ODrive?

ODrive is a powerful brushless motor driver, which is provided with built in software that helps to simplify some tasks as controlling the angular speed, position and current, in addition it comes with a configuration tool in *python 3* that allows to configure properly the board for the motors that would be used, also configure the input from the encoders, it can control a maximum of 2 motors at the same time.
It has some protocols of communication that makes easier the control of the motors. In order to control the board it will be needed a communication between the speed control process and the board ODrive. There are different communication options, as *CAN* and *USB*. This section will cover the type of message and structure in USB mode to command the velocity and request the encoder feedback.
To communicate the board with the computer through the USB method the proposed system by *ODrive robotics* is based on sending an ASCII string that has the followings formats:
Command To modify the velocity of the wheel the next structure will be needed:

$$[v \ \ motor \ \ velocity \ \ current]$$

Where:
v stands for velocity, which is the parameter that will control.
motor the motor that will control, can be 0 or 1.
velocity the angular speed of the wheel in count/s
current is the current feed-forward term, in A which is optional.
After the command is sent there won't be feedback from the ODrive.
Feedback If the system needs to know the actual velocity of a wheel or its position the command is:

$$[f \ \ motor]$$

Where:
f stands for feedback.
motor the motor that will control, can be 0 or 1.
Immediately after the the command is sent the *ODrive* will respond with the *position* followed by the *velocity* of the requested wheel in counts and counts/s
Even the units are far to be international in the section 4.6 will be more detail of how to convert them in International Units.

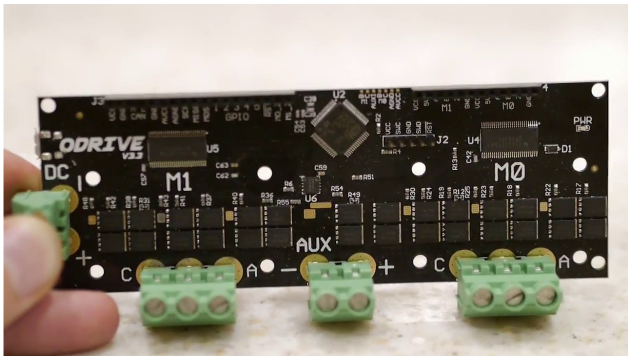


Figure 4.1: ODrive board

4.2 Brushless motors

A brushless motor is a direct current (DC) electric motor that operates without the mechanical brushes and commutator of a traditional brushed motor. A controller drive can provide pulses of current to the motor coils in order to control the speed and torque of the motor.

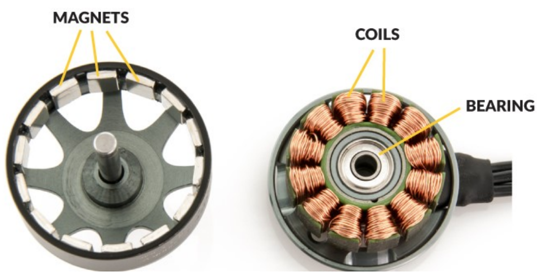
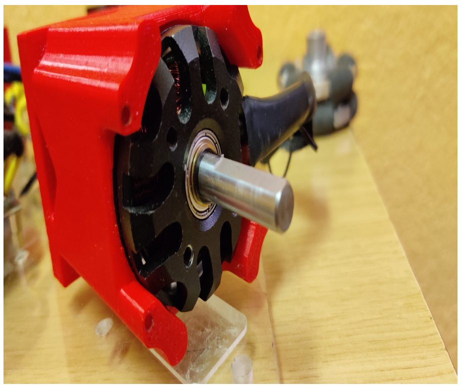
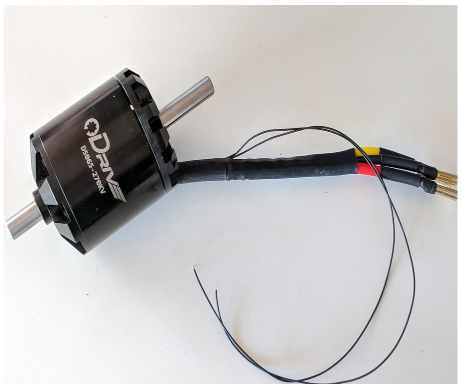


Figure 4.2: Brushless motor

The chosen motors are brushless D5065 270KV. This kind of motors have high torque even without reduction (1.99N.m), at the maximum current which is 65A, the *270KV* is a motor constant which specifies the amount of rpm per volt that can achieve the motor without load, in this case the robot uses a 14.8V battery which means that can spin at $270 \times 14.8 = 3996rpm$. Even though all the components are physically able to manage that amount of current and speed, for safety and durability purposes, the current is limited to 60A as well as the rpm to 318.27 as a result the brute power per motor will be lowered to 50W, the equations are specified below.



(a) Motor D5065 with cage



(b) Motor D5065

The power of the motors can be calculated with the following expressions:

$$P_{mec} = M \times n \times \frac{2\pi}{60} \tag{4.1}$$

Where:
P_{mec} = Power in W
M = Torque in Nm
n = Speed in rpm

The maximum power that the motors can achieve at 32V where the rpm are 8640 and the torque is about 1.99 Nm

$$P_{mec} = 1.99 \times 8640 \times \frac{2\pi}{60}$$

$$P_{mec} \approx 1800W$$

The main restriction applied to the motor is that it can't go faster than 2m/secs as it could be dangerous for the competition, so in order to calculate the power of the motors first is needed to know the speed of the wheels in rpm.

$$\omega = \frac{v}{r} \tag{4.2}$$

Where:
ω = angluar speed in sec^{-1}
v = lineal speed in m/s
r = radius of the wheel in m

$$w = \frac{2}{0.03} = 33.333\frac{rad}{sec}$$

Converted to $\frac{rad}{min}$

$$w = 33.333\frac{rad}{s} \times \frac{60s}{1min} \times \frac{1rev}{2\pi rad} \approx 318.27\frac{rev}{min}$$

The imposed torque is 1.5 N.m which will give the power of one motor.

$$P_{mec} = 1.5 \times 318.27 \times \frac{2\pi}{60}$$

$$P_{mec} \approx 50W$$

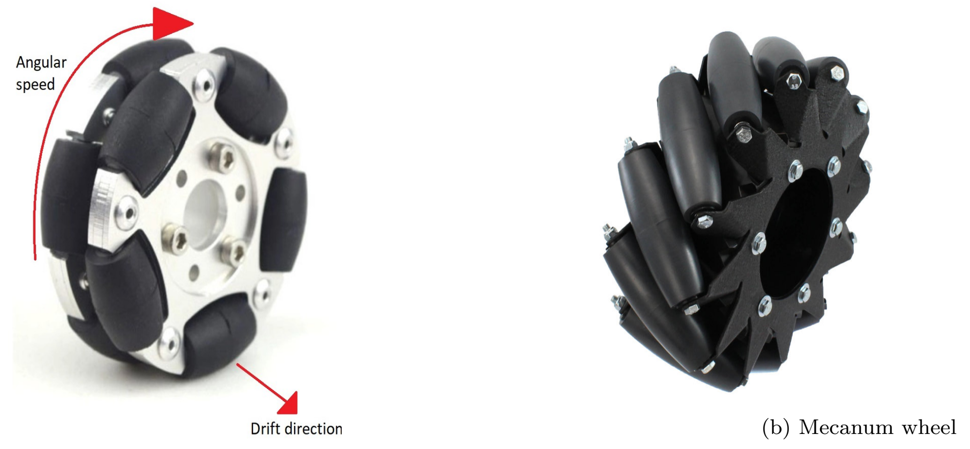
If the *Garrinator* have 4 motors the total power should be 4×50w giving as a result 200w, there is no specified efficiency value in the data sheets so to calculate the electrical power needed won't be possible. However mechanical power gives us an order of magnitude and a lower bound that the electrical system should provide.

4.3 Incremental encoder

It is a linear or rotary electromechanical sensor that has two output signals (A and B), on every change of position will trigger the outputs, depending on the speed will vary the frequency of the input signal, also depending on the order, if it is triggered A first then B implies the sensor is spinning one way, if the order is B-A means that is spinning otherwise.
This sensor does not indicate position, only angular speed and turning direction. In the *Garrinator* robot, encoders are integrated with motors, and encoder data is obtained as explained in the feedback command of ODrive

4.4 Omni wheels

This kind of wheels have small rollers around the circumference which are perpendicular to the turning direction, this provides an effect that the wheel can be driven with full force forward and backward, but will also be capable of slide without mechanical constraint. Do not confuse them with the *Mecanum*, since those wheels have the rollers disposed at 45º from the rim plane, while *omni wheel* have them at 90º.

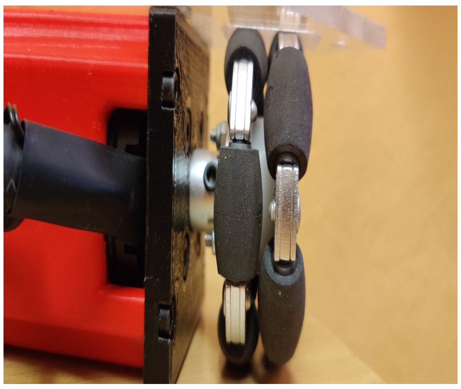


(a) Omni wheel (b) Mecanum wheel

Figure 4.4: Types of holonomic wheels

The chosen wheels for the *Garrinator* are 60 mm diameter and **omnidirectional** type. They are chosen to provide *Garrinator* with 3 degrees of freedom regarding motion (forward/backward, rotation and sideways), providing the *Garrinator* with maximum manoeuvrability as possible inside a 2D plane. Its diameter has to be at least 60mm as the motors are 50mm in diameter, this will give the platform a low profile which won't allow small pieces go inside or below the motors.

(a) Omni wheel front view


(b) Omni wheel left view

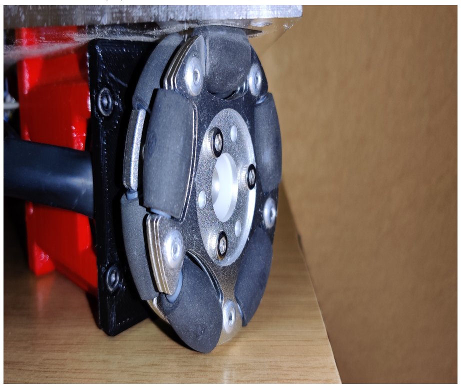

(c) Omni wheel iso view

Figure 4.5: *Garrinator* wheels

4.5 Battery

Figure 4.6: ODrive board

The battery is a Li-PO type, which offers 14.8V and have 5200mAh with a 50c of discharge rate. The discharge rate is a multiplying factor with the capacity, this operation reveals the amount of current that the battery can give as discharge peak. In this case, it means that this battery can offer $5.2A \times 50 = 260A$. With this value the maximum electrical power that can give "safely" is $14.8V \times 260A = 3700w$. In any case this robot is a development project and it is not meant to have a large autonomy, with the parameters acquired in the section *motors* the motors should be able to run 1.38min at full throttle.

$$4motors \times 60A = 240A$$

$$\frac{5.2A/h}{240} = 0.0216h = 78sec$$

4.6 ODrive

The ODrive is one of the angular stones in this project. It is the component that will process the velocity requests from the system and be responsible to give the feedback in time with great accuracy in both actions. It has been selected because its great performance, great reviews, and multiple configuration options. Nevertheless the documentation provided describing the communication protocol is quite poor and the units that uses are far to be international.
This section will bring some light to work with international units.
As seen in the previous chapter the message to move one motor is:

$$[v \qquad M \qquad velocity]$$

v= velocity command
M= motor to move (0 or 1)
velocity= number of counts/secs
The encoders used in this project are configured with 512 $\frac{pulses}{revolution}$ there is a relation between pulses per revolution and counts per revolution.

$$1\frac{pulses}{revolution} = 4\frac{count}{revolution}$$

applied to the encoders:

$$512\frac{pulses}{revolution} \times 4 = 2048\frac{count}{revolution}$$

for example if the system need to move the wheel at 10 rad/s the number of counts/s is:

$$10\frac{rad}{s} \times \frac{1\ \ revolution}{2\pi\ \ rad} \times 2048\frac{count}{revolution} = 2259.5\frac{count}{s}$$

The same happens calling the feedback command:

$$[f \qquad M]$$

f= feedback command
M= motor requested (0 or 1)
response = speed of the motor in counts/sec
if the response to the command is $10\frac{count}{s}$ the equivalent in rad/s is:

$$10\frac{count}{s} \times \frac{revolution}{2048\ \ counts} \times \frac{2\pi\ \ rad}{1\ \ revolution} = 0.0306\frac{rad}{s}$$

4.7 Computer

The computer is a NUC7i7DNH which means that has an *intel i7* processor. The main characteristic is that it is compact and powerful. The Operating system chosen is Ubuntu 18.04 LTS (Long Term Support) the reason of that is because ROS runs on aLinux based environment.
The ROS version is *Melodic Morenia* which is the latest ROS version recommended.

To finalize the chapter, all the connections between the components are shown in the figure 4.7.

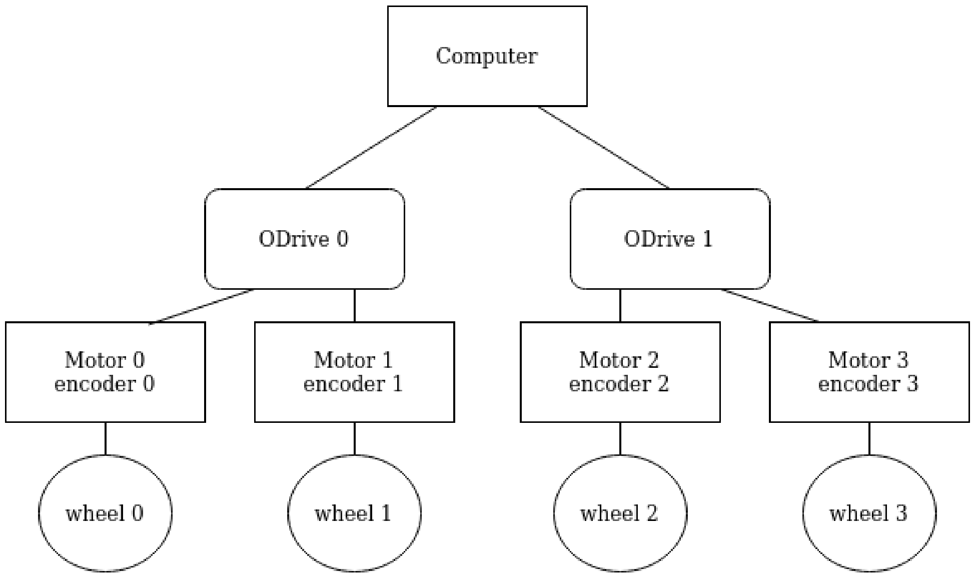


Figure 4.7: Hardware structure

The *Garrinator* is a mobile robot property of the robot team from University of Vic, the robot competes every year in different challenges, where there are some restrictions, as a dimension (max. 300 x 300 mm), power 24V max. and some more. The robotic team has some sponsors that help economically to purchase the new components in order to adapt the *Garrinator* to the new challenge, that means that another building restriction is the price of the components as the budget is limited. Given those restrictions the team aims to find the best components as possible.

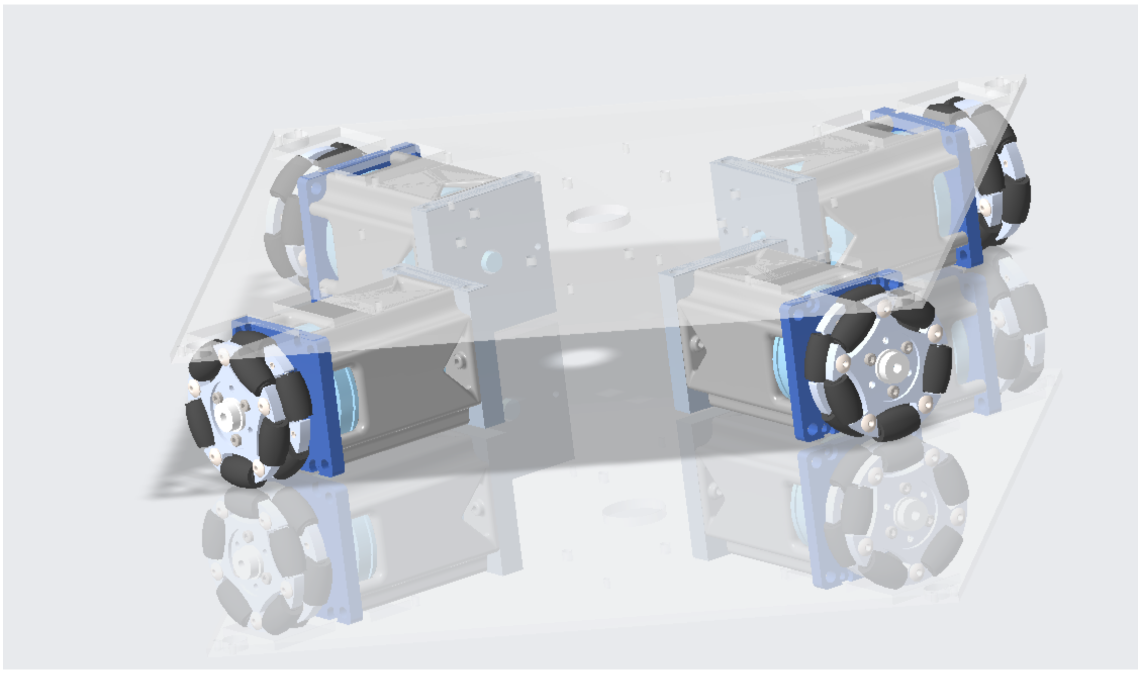


Figure 4.8: Garrinator modeled in Creo parametric

Chapter 5

Software architecture

This chapter will cover the main concepts needed to understand the project and its functionalities, the major concepts of ROS are inspired by the book *Mastering ROS for Robotics Programming by Jonathan Cacace*

5.1 What is ROS?

ROS *Robot Operating System* is a free, open-source, high-level operating system focused on robots. It offers hardware abstraction, low level device control, communication between processes and package management. In addition it has multiple tools and libraries for building, writing and running code through multiple computers and code reusing.

5.1.1 Why use ROS and its benefits

High-end capabilities: ROS comes with ready-to-use capabilities. For example, Simultaneous Localization and Mapping (SLAM) and Adaptive Monte Carlo Localization (AMCL) packages in ROS can be used for performing autonomous navigation in mobile robots. These capabilities are its best form of implementation, so writing new code for existing capabilities is like reinventing the wheel.

Tons of tools: ROS has tools that help to debug, visualize and simulate the robot the more common ones are Rviz and Gazebo.

Support for high-end sensors and actuators: ROS is packed with device drivers and interface packages of various sensors and actuators in robotics. The high-end sensors include Laser scanners, Kinect, and so on.

Modularity: When codding a robot with standalone applications may cause that if any of the main threads crashes the whole robot can stop, but in ROS there are different nodes for each process and if the node crashes, the system can still work, in addition ROS provides robust methods to resume operations even if any sensors or motors are dead.

5.1.1.1 Nodes

Nodes are the processes that perform computations. Each ROS node is written using ROS client libraries, it can be written using different languages as **C++**, and **Python**. Also, it can implement different ROS functionalities, such as communication between nodes, which is particularly useful when different nodes of the robot must exchange information between them.
One of the aim of ROS nodes is to build simpler processes rather than a large process with all the functionality. Being a simple structure, ROS nodes are easy to debug.

Messages Nodes communicate with each other using messages. Messages are simply a data structure containing the typed field, which can hold a set of data, and that can be sent to another node.

Topic All the messages in ROS are transported through buses that are called topics, they have a unique name that identifies it. A node can either **subscribe** in order to receive the information flowing in the bus or **publish** a message, so any node subscribed to this topic will receive it.

5.1.1.2 Services

The services that provide ROS are request/response between *nodes*. One node can ask for the execution of a fast procedure to another node; for example, asking for some quick calculation

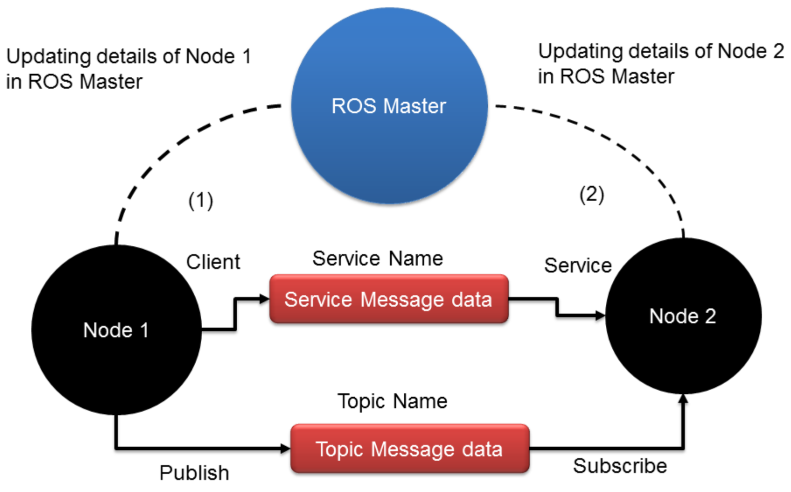


Figure 5.1: Communication diagram between nodes

5.2 Garrinator nodes

This section will cover the ROS nodes, messages and main structure used in this project as well as how they are configured and interact with each other. Is great to mention that all the nodes must have the established input/output in order to be a modular system, so if in the future there will be any change to one or multiple nodes in the main system is preserved as is, and only is needed to modify the desired node.

5.2.1 Hardware interface

The hardware interface is in charge of moving the actuators and giving feedback of its status at any moment. In this case it would be the 4 motors attached to 2 *ODrive*.It is also needed that the units of both actions mentioned above are in standard units. The functions that have to be implemented are **read()**, **write()**. These two methods are the interface through the real robot and the system. In order to control the wheels they are declared as a joint of one of the three joint interface types: position, velocity and effort. The project opted for velocity as it would simplify some calculus and the *ODrive* is commanded and provides this velocity data, with the conversions explained before.

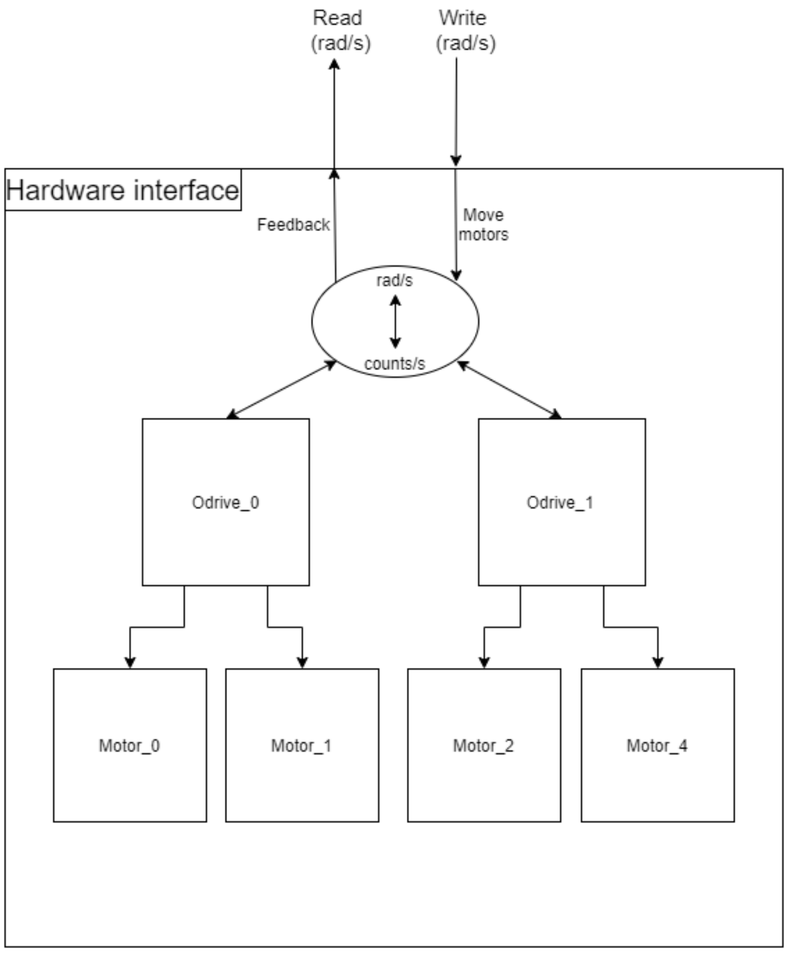


Figure 5.2: Hardware interface diagram

5.2.2 Controller

The controller is the middle process in the teleop chain. It will receive the twist generated from the teleop node by subscribing to that topic `robot/cm_vel` and whenever its values are updated will call a function (callback) that will compute the new wheel angular velocities in order to satisfy that new robot state. This computation is done with the inverse kinematics model explained in chapter 3. Once there are computed, they are sent to the hardware interface to apply the changes.

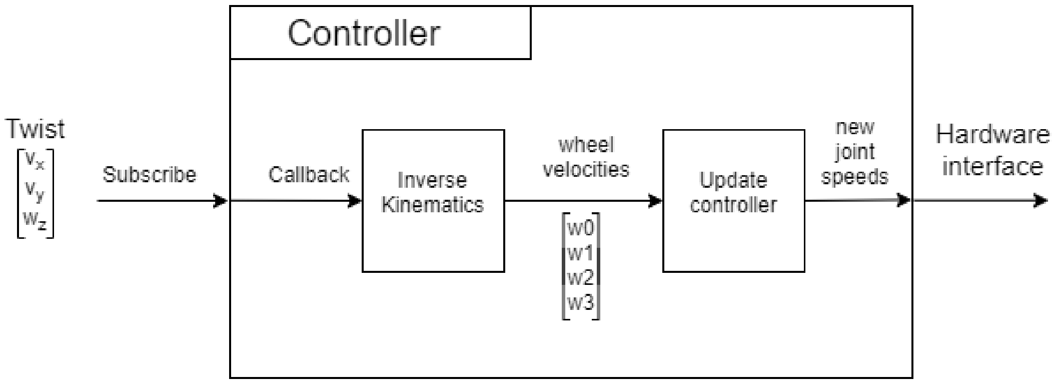


Figure 5.3: Controller diagram

5.2.3 Teleop

This node will be in charge to receive the state of the joystick, every time it have a different position. With the new joystick state, it computes a new robot *tiwst* to the system. The values that give the joystick are between 1 and -1, by multiplying a constant 2 will ensure that the max value that the system is able to produce is 2m/s. Furthermore as a joystick have multiple buttons and two sticks, it is configured that the left stick will produce the components v_x and v_y while the right one does the w_z.

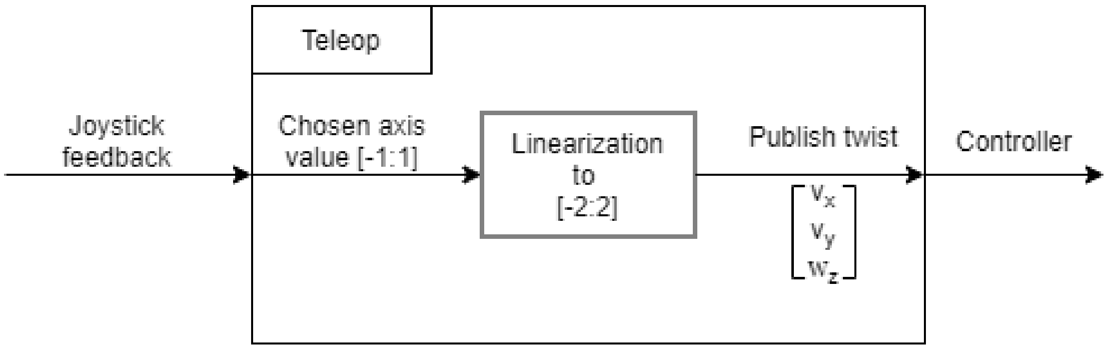


Figure 5.4: Teleop node diagram

To conclude this chapter the connection between the nodes could be simplified as the figure 5.4. Even though it could look quite simple, its complexity is outstanding for a newbie, even more when all the parameters, functions and protocols are quite ambiguous and with lack of practical examples.

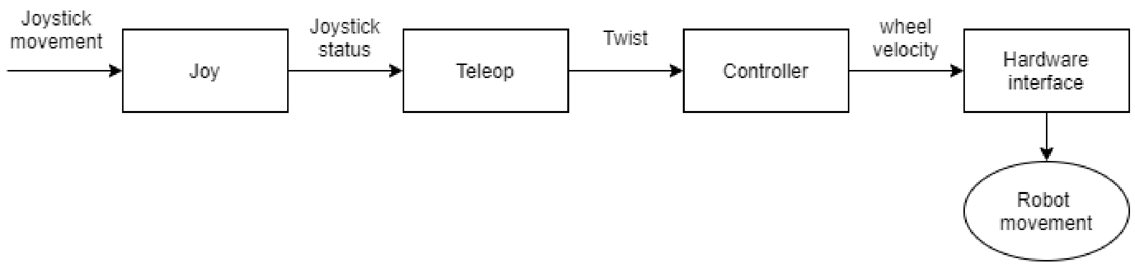
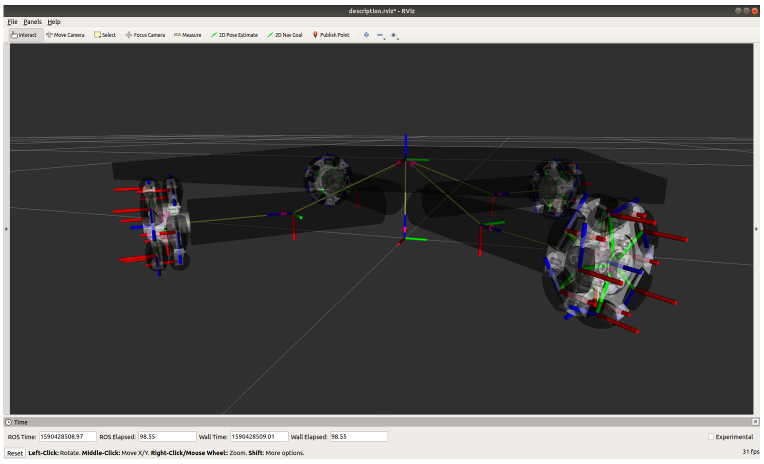


Figure 5.5: Garrinator ROS structure

Chapter 6

Results

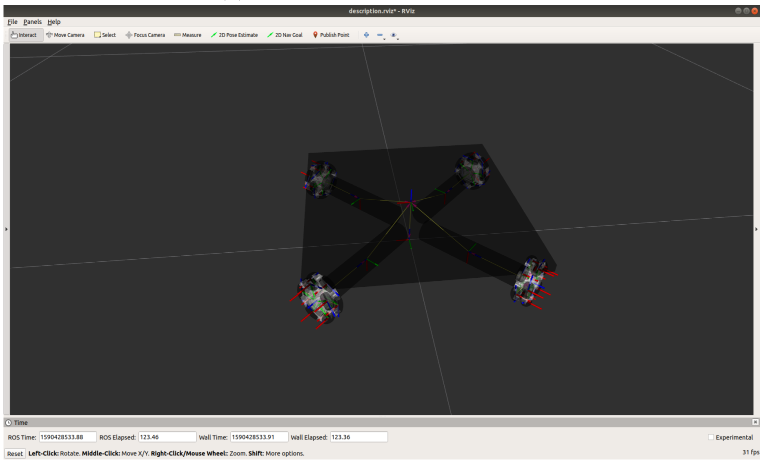
The figure 6.1 shows the *Garrinator* modeled with URDF files, which are the ones that use ROS, also it is possible to observe all the coordinates axis of every component, the issues happened here where wrong magnitude units of the stl files, the URDF ROS interpreter reads the units as meters, while they were meant to be mm. After some corrections the proportions of the *Garrinator* went back to normal.



ROS Time: 1590428508.97 ROS Elapsed: 98.55 Wall Time: 1590428509.01 Wall Elapsed: 98.55 Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options. 31 fps

(a) *Garrinator* URDF front view



ROS Time: 1590428533.88 ROS Elapsed: 123.46 Wall Time: 1590428533.91 Wall Elapsed: 123.36 Experimental

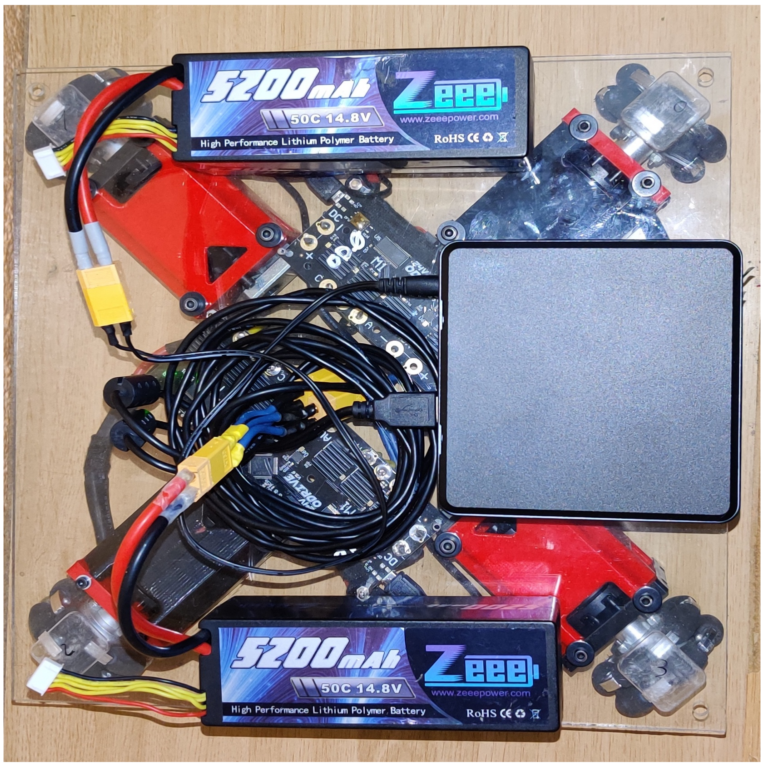
Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options. 31 fps

(b) *Garrinator* URDF top view

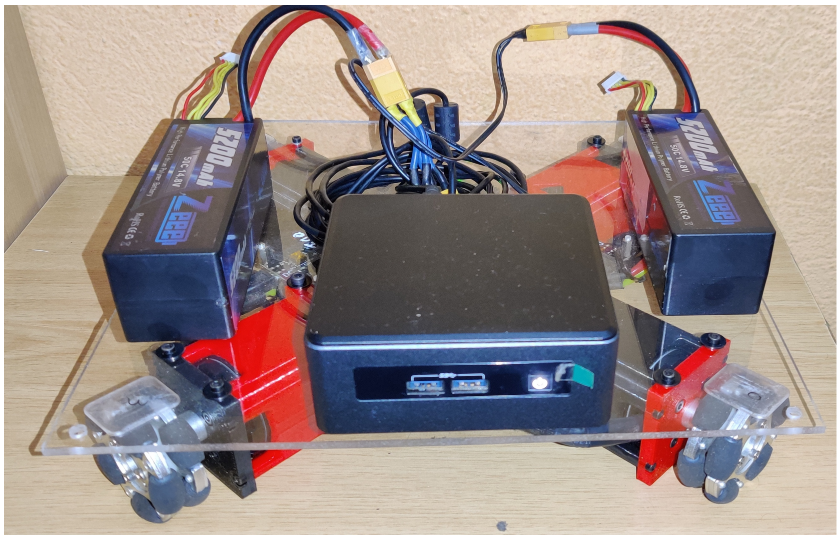
Figure 6.1: *Garrinator* URDF

Other issues that appeared developing the project were, the difficulty to write an appropriate hardware interface node, send an ASCII string through the USB is more complex in Linux than in other Operating systems. As it is needed to open and configure the port in order to start transmitting commands.

The final physical build of the *Garrinator* can be appreciated in fig 6.2



(a) *Garrinator* physical front view



(b) *Garrinator* physical top view

Figure 6.2: *Garrinator* final build

In order to startup the system it will be needed the next procedure: Plug-in the battery to the ODrive power connector and let them calibrate. Plug-in the other battery to the *Garrinator* computer and turn it on. Later connect the computer and the user computer to the same wi-fi network, open an ssh terminal and establish connection from user computer to the *Garrinator*, this will allow the user access to the Linux terminal of the robot. Turn on the joystick and wait until the middle led stay still this means there is Bluetooth connection between the robots computer and the joystick. Launch the package *Omni4 Controller* with the Linux terminal and wait until it finish loading the controller. At this point the *Garrinator* is ready to be teleoperated.

Chapter 7

Conclusions

The conclusion of this project is that all the objectives have been achieved, the power train of the mobile robot is formed by 4 omniwheels with brushless motors, everything is programmed inside the Robot Operating System (ROS), using the language *C++*, and the last but not least the robot is able to be teleoperated with a joystick via Bluetooth. Leading a system which is highly modular, because all the components of software and hardware are now just little modules (ROS nodes), that can be replaced if they are needed, but the system won't be affected, also it is robust as the ROS system can continue functioning even if some nodes malfunction. On the other hand this project has been a challenge from the beginning, but with hard work and guidance from the tutors its been possible, personally I've enjoyed doing it, sometimes it could be somewhat frustrating, but by dealing with errors and learning from them, I've learnt a lot of the ROS basis, also is good to mention that ROS has a quite sharp learning curve so it is more difficult at the beginning than at the end. I'm proud of the results achieved.

Bibliography

Asimov Robotics. URL https://web.archive.org/web/20190818064119/http://www.asimovrobotics.com:80/{#}section-2.

Oxford Learner's Dictionaries | Find definitions, translations, and grammar explanations at Oxford Learner's Dictionaries. URL https://www.oxfordlearnersdictionaries.com/.

Lentin Joseph_ Jonathan Cacace - Mastering ROS for Robotics Programming - Second Edition_ Design, build, and simulate complex robots using the Robot Operating System-Packt Publishing (2018).pdf - Google Play. URL https://play.google.com/books/reader?id=Ui0fJwAAAEAJ{&}pg=GBS.PA33.

Robots/Mecanumbot - ROS Wiki. URL http://wiki.ros.org/Robots/Mecanumbot.

DC Motor Tutorial - DC Motor Calculation. URL https://www.faulhaber.com/en/support/technical-support/motors/tutorials/dc-motor-tutorial-dc-motor-calculation/.

Getting Started | ODrive, a. URL https://docs.odriverobotics.com/.

ASCII Protocol | ODrive, b. URL https://docs.odriverobotics.com/ascii-protocol.html{#}motor-trajectory-command.

Introduction to ROS - Learning Robotics using Python - Second Edition, a. URL https://subscription.packtpub.com/book/hardware{_}and{_}creative/9781788623315/1/ch01lvl1sec11/introduction-to-ros.

ROS/Introduction - ROS Wiki, b. URL http://wiki.ros.org/ROS/Introduction.

ROS/Tutorials - ROS Wiki, c. URL http://wiki.ros.org/ROS/Tutorials.

Seekur mobile robot - Génération Robots. URL https://www.generationrobots.com/en/402404-robot-mobile-seekur.html.

geometry_msgs/Twist Documentation. URL http://docs.ros.org/melodic/api/geometry{_}msgs/html/msg/Twist.html.

Launch Files — ROS Tutorials 0.5.1 documentation. URL http://www.clearpathrobotics.com/assets/guides/kinetic/ros/LaunchFiles.html.

nav2™ - CrossWing Inc., a. URL http://crosswing.com/nav2/.

Robots/Nav2 - ROS Wiki, b. URL http://wiki.ros.org/Robots/Nav2.

A. corominas Murtra. andreucm/essential_maths_roboticists: Essential Maths for Roboticists. URL `https://github.com/andreucm/essential{_}maths{_}roboticists`.

Appendix A

Code Git repository

The code used in this project is uploaded in the repository of the GitHub platform, following the next structure:

- Description package
  - Launch folder with launch file.
  - Macros of the wheels with the 10 rollers.
  - Mesh folder with STL files.
  - Rviz configuration.
- Garri teleop package.
  - Launch folder with launch file.
  - Source folder with node code.
- Hardware interface package.
  - Config folder with yaml configuration file.
  - Launch folder with launch file.
  - folder with header files.
  - Source folder with the node and the hardware interface code.
- Omni 4 controller package
  - Config folder with yaml configuration file.
  - Launch folder with launch file.
  - folder with header files.
  - Source folder with the node and the hardware interface code.
  - Controller plugin file to save the library.

Link: `https://github.com/JavierRubia/Garrinator`